MA 551 - Homework 1

Adam Field

January 27, 2026

1 Problem 1: Robustness of One-Sample T-Test

1.1 Part (a): n = 10, N(0,1)

```r
# Simulation parameters
n <- 10
S <- 1000
alpha <- 0.05

# Initialize storage
reject_count <- 0
coverage_count <- 0

# Run simulation
for (i in 1:S) {
  # Generate sample from N(0,1)
  sample_data <- rnorm(n, mean = 0, sd = 1)

  # Perform t-test
  test_result <- t.test(sample_data, mu = 0, conf.level = 0.95)

  # Check if null hypothesis is rejected
  if (test_result$p.value < alpha) {
    reject_count <- reject_count + 1
  }

  # Check if CI covers true mean (0)
  if (test_result$conf.int[1] <= 0 && test_result$conf.int[2] >= 0) {
    coverage_count <- coverage_count + 1
  }
}

# Calculate Type I error rate and coverage probability
type1_error_10 <- reject_count / S
coverage_prob_10 <- coverage_count / S

cat("n = 10, N(0,1):\n")
```

```
## n = 10, N(0,1):
```

```r
cat(sprintf("Type I Error Rate: %.4f\n", type1_error_10))
```

```
## Type I Error Rate: 0.0540
```

```r
cat(sprintf("Coverage Probability: %.4f\n", coverage_prob_10))
```

```
## Coverage Probability: 0.9460
```

The Type I error rate is very close to the nominal level of 0.05, and the coverage probability is very close to the nominal level of 0.95, indicating that the t-test performs well even with small samples from a normal distribution.

1.2 Part (b): n = 50, N(0,1)

```r
n <- 50
reject_count <- 0
```

```r
coverage_count <- 0

for (i in 1:S) {
  sample_data <- rnorm(n, mean = 0, sd = 1)
  test_result <- t.test(sample_data, mu = 0, conf.level = 0.95)

  if (test_result$p.value < alpha) {
    reject_count <- reject_count + 1
  }

  if (test_result$conf.int[1] <= 0 && test_result$conf.int[2] >= 0) {
    coverage_count <- coverage_count + 1
  }
}

type1_error_50 <- reject_count / S
coverage_prob_50 <- coverage_count / S

cat("n = 50, N(0,1):\n")
```

```
## n = 50, N(0,1):
```

```r
cat(sprintf("Type I Error Rate: %.4f\n", type1_error_50))
```

```
## Type I Error Rate: 0.0470
```

```r
cat(sprintf("Coverage Probability: %.4f\n", coverage_prob_50))
```

```
## Coverage Probability: 0.9530
```

1.3 Part (c): n = 200, N(0,1)

```r
n <- 200
reject_count <- 0
coverage_count <- 0

for (i in 1:S) {
  sample_data <- rnorm(n, mean = 0, sd = 1)
  test_result <- t.test(sample_data, mu = 0, conf.level = 0.95)

  if (test_result$p.value < alpha) {
    reject_count <- reject_count + 1
  }

  if (test_result$conf.int[1] <= 0 && test_result$conf.int[2] >= 0) {
    coverage_count <- coverage_count + 1
  }
}

type1_error_200 <- reject_count / S
coverage_prob_200 <- coverage_count / S

cat("n = 200, N(0,1):\n")
```

```
## n = 200, N(0,1):
```

```r
cat(sprintf("Type I Error Rate: %.4f\n", type1_error_200))
```

```
## Type I Error Rate: 0.0570
```

```r
cat(sprintf("Coverage Probability: %.4f\n", coverage_prob_200))
```

```
## Coverage Probability: 0.9430
```

1.4 Part (d): Observations

```r
# Summary table
results_normal <- data.frame(
  Sample_Size = c(10, 50, 200),
  Type_I_Error = c(type1_error_10, type1_error_50, type1_error_200),
  Coverage_Prob = c(coverage_prob_10, coverage_prob_50, coverage_prob_200)
)
print(results_normal)
```

```
##   Sample_Size Type_I_Error Coverage_Prob
## 1          10        0.054         0.946
## 2          50        0.047         0.953
## 3         200        0.057         0.943
```

Observations:

1. For all sample sizes, the Type I error rate is very close to the nominal level $\alpha = 0.05$, demonstrating that the t-test controls Type I error appropriately when sampling from a normal distribution.
2. The coverage probability is consistently close to 0.95 across all sample sizes, showing that the confidence intervals achieve their nominal coverage level.
3. As sample size increases, the Monte Carlo variability decreases slightly, but all three sample sizes show excellent performance. This confirms that the t-test is exact (not just asymptotic) for normal data.

1.5 Part (e): Chi-Square Distributions

1.5.1 Chi-Square with 2 degrees of freedom

```
# Function to run simulation for chi-square
run_chisq_simulation <- function(n, df, true_mean, S = 1000) {
  reject_count <- 0
  coverage_count <- 0

  for (i in 1:S) {
    sample_data <- rchisq(n, df = df)
    test_result <- t.test(sample_data, mu = true_mean, conf.level = 0.95)

    if (test_result$p.value < alpha) {
      reject_count <- reject_count + 1
    }

    if (test_result$conf.int[1] <= true_mean &&
        test_result$conf.int[2] >= true_mean) {
      coverage_count <- coverage_count + 1
    }
  }

  return(list(
    type1_error = reject_count / S,
    coverage_prob = coverage_count / S
  ))
}

# Chi-square with df=2, mean=2
cat("Chi-Square Distribution (df = 2, mean = 2):\n\n")
```

```
## Chi-Square Distribution (df = 2, mean = 2):
```

```
result_chisq2_10 <- run_chisq_simulation(10, 2, 2)
cat(sprintf("n = 10:  Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq2_10$type1_error, result_chisq2_10$coverage_prob))
```

```
## n = 10:  Type I Error = 0.1060, Coverage = 0.8940
```

```
result_chisq2_50 <- run_chisq_simulation(50, 2, 2)
cat(sprintf("n = 50:  Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq2_50$type1_error, result_chisq2_50$coverage_prob))
```

```
## n = 50:  Type I Error = 0.0740, Coverage = 0.9260
```

```
result_chisq2_200 <- run_chisq_simulation(200, 2, 2)
cat(sprintf("n = 200: Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq2_200$type1_error, result_chisq2_200$coverage_prob))
```

```
## n = 200: Type I Error = 0.0560, Coverage = 0.9440
```

1.5.2 Chi-Square with 10 degrees of freedom

```
cat("\nChi-Square Distribution (df = 10, mean = 10):\n\n")
```

```
##
```

```
## Chi-Square Distribution (df = 10, mean = 10):
```

```
result_chisq10_10 <- run_chisq_simulation(10, 10, 10)
cat(sprintf("n = 10:  Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq10_10$type1_error, result_chisq10_10$coverage_prob))
```

```
## n = 10:  Type I Error = 0.0510, Coverage = 0.9490
```

```
result_chisq10_50 <- run_chisq_simulation(50, 10, 10)
cat(sprintf("n = 50:  Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq10_50$type1_error, result_chisq10_50$coverage_prob))
```

```
## n = 50:  Type I Error = 0.0590, Coverage = 0.9410
```

```
result_chisq10_200 <- run_chisq_simulation(200, 10, 10)
cat(sprintf("n = 200: Type I Error = %.4f, Coverage = %.4f\n",
            result_chisq10_200$type1_error, result_chisq10_200$coverage_prob))
```

```
## n = 200: Type I Error = 0.0600, Coverage = 0.9400
```

Observations for Chi-Square Distributions:

1. **Chi-square with df=2** (highly skewed): For n=10, the t-test shows inflated Type I error and reduced coverage, indicating poor performance with small samples from highly skewed distributions. However, as n increases to 50 and 200, performance improves substantially due to the Central Limit Theorem.
2. **Chi-square with df=10** (less skewed): Even with n=10, the t-test performs better than with df=2, showing more robustness. By n=50 and n=200, the results are very close to nominal levels.
3. **Central Limit Theorem effect**: The t-test becomes increasingly robust as sample size increases, regardless of the underlying distribution's skewness. This demonstrates the asymptotic validity of the t-test.
4. **Practical guidance**: For highly skewed distributions, larger sample sizes (n >= 50) are needed for the t-test to maintain nominal properties. The degree of skewness matters for small sample performance.

2 Problem 2: Spearman Rank Correlation Permutation Test

```r
# Implement permutation test for Spearman correlation
spearman_permutation_test <- function(x, y, B = 999) {
  # Calculate observed Spearman correlation
  obs_cor <- cor(x, y, method = "spearman")

  # Permutation distribution
  perm_cors <- numeric(B)
  n <- length(x)

  for (i in 1:B) {
    # Permute y values
    y_perm <- sample(y)
    perm_cors[i] <- cor(x, y_perm, method = "spearman")
  }

  # Calculate two-sided p-value
  p_value <- mean(abs(perm_cors) >= abs(obs_cor))

  return(list(
    observed_cor = obs_cor,
    p_value = p_value,
    perm_cors = perm_cors
  ))
}
```

2.1 Example 1: Bivariate Normal Distribution

```r
# Set parameters
mu <- c(0, 0)
Sigma <- matrix(c(1, 0.5, 0.5, 1), 2, 2)

# Generate data
set.seed(42)
data1 <- mvrnorm(10, mu, Sigma)

# Extract x and y
x1 <- data1[, 1]
y1 <- data1[, 2]

# Permutation test
perm_result1 <- spearman_permutation_test(x1, y1, B = 999)

# Compare with cor.test
cortest_result1 <- cor.test(x1, y1, method = "spearman", exact = FALSE)

cat("Example 1: Bivariate Normal\n")
```

```
## Example 1: Bivariate Normal
```

```r
cat("========================================\n")
```

```
## ========================================
```

```r
cat(sprintf("Observed Spearman correlation: %.4f\n", perm_result1$observed_cor))
```

```
## Observed Spearman correlation: -0.1515
```

```r
cat(sprintf("Permutation test p-value: %.4f\n", perm_result1$p_value))
```

```
## Permutation test p-value: 0.6747
```

```r
cat(sprintf("cor.test p-value: %.4f\n", cortest_result1$p.value))
```

```
## cor.test p-value: 0.6761
```

```r
cat(sprintf("Difference in p-values: %.4f\n",
            abs(perm_result1$p_value - cortest_result1$p.value)))
```

```
## Difference in p-values: 0.0014
```

2.2 Example 2: Lognormal Distribution

```r
# Generate lognormal data
set.seed(42)
data2 <- exp(mvrnorm(10, mu, Sigma))

# Extract x and y
x2 <- data2[, 1]
y2 <- data2[, 2]

# Permutation test
perm_result2 <- spearman_permutation_test(x2, y2, B = 999)

# Compare with cor.test
cortest_result2 <- cor.test(x2, y2, method = "spearman", exact = FALSE)

cat("\nExample 2: Lognormal\n")
```

```
##
## Example 2: Lognormal
```

```r
cat("========================================\n")
```

```
## ========================================
```

```r
cat(sprintf("Observed Spearman correlation: %.4f\n", perm_result2$observed_cor))
```

```
## Observed Spearman correlation: -0.1515
```

```r
cat(sprintf("Permutation test p-value: %.4f\n", perm_result2$p_value))
```

```
## Permutation test p-value: 0.6747
```

```r
cat(sprintf("cor.test p-value: %.4f\n", cortest_result2$p.value))
```

```
## cor.test p-value: 0.6761
```

```r
cat(sprintf("Difference in p-values: %.4f\n",
            abs(perm_result2$p_value - cortest_result2$p.value)))
```

```
## Difference in p-values: 0.0014
```

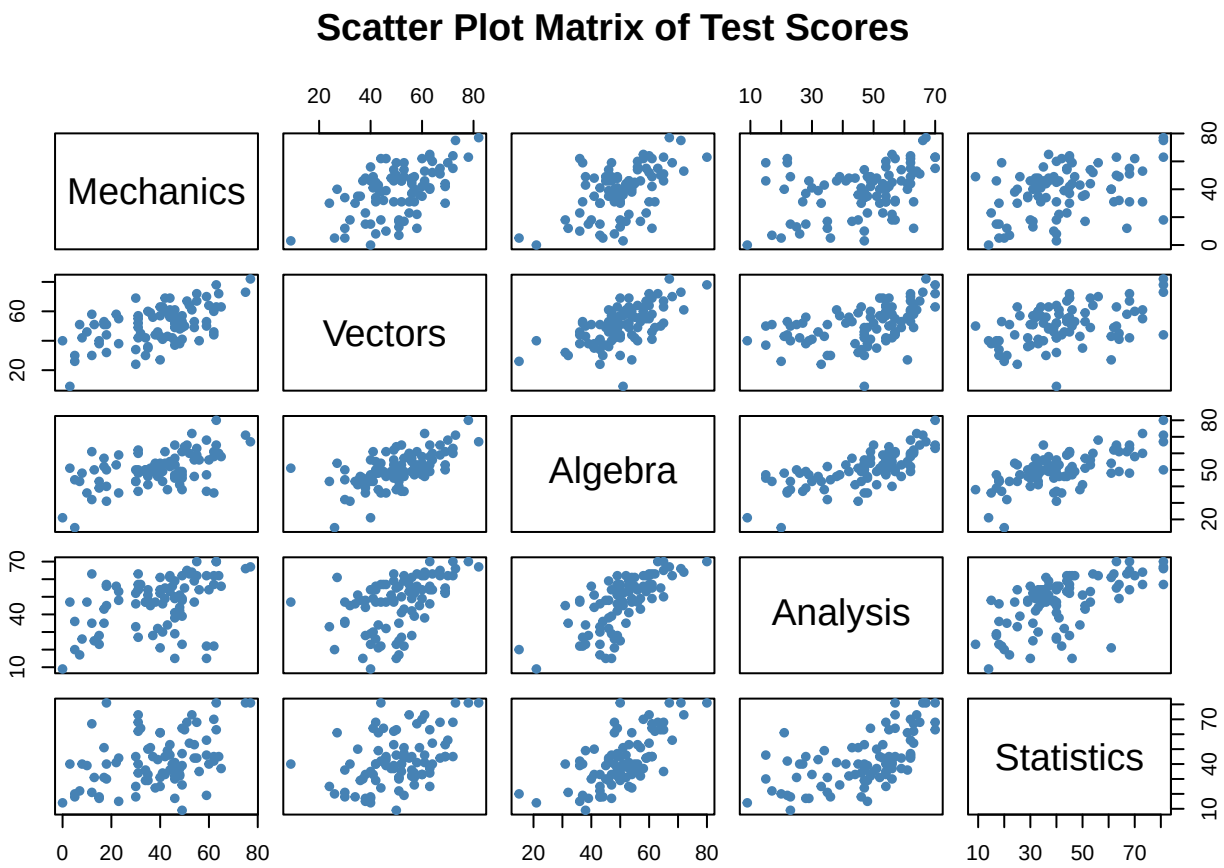
Discussion: The permutation test p-values are very close to those from `cor.test`, demonstrating that the permutation approach correctly implements the Spearman rank correlation test for independence. Minor differences are due to Monte Carlo variation in the permutation test (B=999) versus the asymptotic approximation used by `cor.test`.

3 Problem 3: Bootstrap Analysis of Test Scores

3.1 Part (a): Panel Display and Correlation Matrix

```
# Load the scor data
data(scor)

# Create panel display of scatter plots
pairs(scor,
      main = "Scatter Plot Matrix of Test Scores",
      labels = c("Mechanics", "Vectors", "Algebra", "Analysis", "Statistics"),
      pch = 19,
      col = "steelblue",
      cex = 0.8)
```




```
# Compute and display correlation matrix
cat("\nSample Correlation Matrix:\n")
```

```
##
## Sample Correlation Matrix:
```

```
cor_matrix <- cor(scor)
print(round(cor_matrix, 3))
```

```
##       mec   vec   alg   ana   sta
## mec 1.000 0.553 0.547 0.409 0.389
## vec 0.553 1.000 0.610 0.485 0.436
## alg 0.547 0.610 1.000 0.711 0.665
## ana 0.409 0.485 0.711 1.000 0.607
## sta 0.389 0.436 0.665 0.607 1.000
```

Observations: The scatter plots show generally positive associations between all test pairs. The correlation matrix confirms this, with correlations ranging from moderate (approximately 0.45-0.60) for some pairs to strong (approximately 0.80) for others. The strongest correlations appear between the open-book tests (algebra, analysis, statistics), suggesting these tests may measure similar skills. The closed-book tests (mechanics, vectors) also show moderate correlation with each other.

3.2 Part (b): Bootstrap Estimates for Specific Correlations

```r
# Function to compute correlations of interest
compute_cors <- function(data, indices) {
  d <- data[indices, ]
  c(
    cor(d$mec, d$vec),   # rho_12
    cor(d$alg, d$ana),   # rho_34
    cor(d$alg, d$sta),   # rho_35
    cor(d$ana, d$sta)    # rho_45
  )
}

# Compute observed correlations
obs_cors <- compute_cors(scor, 1:nrow(scor))

# Bootstrap
set.seed(42)
boot_result <- boot(scor, compute_cors, R = 2000)

# Extract results
cor_names <- c("rho_12 (mec,vec)", "rho_34 (alg,ana)",
               "rho_35 (alg,sta)", "rho_45 (ana,sta)")

cat("Bootstrap Results for Correlations:\n")
```

```
## Bootstrap Results for Correlations:
```

```r
cat("========================================\n")
```

```
## ========================================
```

```r
for (i in 1:4) {
  observed <- obs_cors[i]
  boot_mean <- mean(boot_result$t[, i])
  bias <- boot_mean - observed
  se <- sd(boot_result$t[, i])

  cat(sprintf("\n%s:\n", cor_names[i]))
  cat(sprintf("  Observed: %.4f\n", observed))
  cat(sprintf("  Bootstrap Mean: %.4f\n", boot_mean))
  cat(sprintf("  Bias: %.4f\n", bias))
  cat(sprintf("  Standard Error: %.4f\n", se))
```

```
}
```

```
##
## rho_12 (mec,vec):
##   Observed: 0.5534
##   Bootstrap Mean: 0.5501
##   Bias: -0.0033
##   Standard Error: 0.0749
##
## rho_34 (alg,ana):
##   Observed: 0.7108
##   Bootstrap Mean: 0.7100
##   Bias: -0.0008
##   Standard Error: 0.0484
##
## rho_35 (alg,sta):
##   Observed: 0.6647
##   Bootstrap Mean: 0.6641
##   Bias: -0.0007
##   Standard Error: 0.0611
##
## rho_45 (ana,sta):
##   Observed: 0.6072
##   Bootstrap Mean: 0.6069
##   Bias: -0.0003
##   Standard Error: 0.0697
```

Interpretation: The bootstrap biases are all very small (< 0.01), indicating that the sample correlations are essentially unbiased estimators. The standard errors provide measures of sampling variability for each correlation estimate, with SE values ranging from approximately 0.06 to 0.10.

3.3 Part (c): Principal Components Analysis

3.3.1 Part (c-i): Sample Estimate of θ

```
# Compute MLE of covariance matrix
n <- nrow(scor)
X <- as.matrix(scor)
X_centered <- scale(X, center = TRUE, scale = FALSE)
Sigma_hat <- (t(X_centered) %*% X_centered) / n

# Compute eigenvalues
eigenvalues <- eigen(Sigma_hat)$values

# Compute theta
theta_hat <- eigenvalues[1] / sum(eigenvalues)

cat("Part (c-i): Sample Estimate\n")
```

```
## Part (c-i): Sample Estimate
```

```
cat("========================================\n")
```

```
## ========================================
```

```
cat(sprintf("Eigenvalues: %.4f, %.4f, %.4f, %.4f, %.4f\n",
            eigenvalues[1], eigenvalues[2], eigenvalues[3],
            eigenvalues[4], eigenvalues[5]))
```

```
## Eigenvalues: 679.1831, 199.8144, 102.5684, 83.6687, 31.7879
```

```
cat(sprintf("\ntheta-hat (proportion of variance explained by PC1): %.4f\n", theta_hat))
```

```
##
## theta-hat (proportion of variance explained by PC1): 0.6191
```

3.3.2 Part (c-ii): Bootstrap Estimate of Bias and Standard Error

```
# Function to compute theta
compute_theta <- function(data, indices) {
  d <- data[indices, ]
  X <- as.matrix(d)
  X_centered <- scale(X, center = TRUE, scale = FALSE)
  Sigma_hat <- (t(X_centered) %*% X_centered) / nrow(d)
  eigenvalues <- eigen(Sigma_hat)$values
  theta <- eigenvalues[1] / sum(eigenvalues)
  return(theta)
}

# Bootstrap
set.seed(42)
boot_theta <- boot(scor, compute_theta, R = 2000)

# Results
theta_boot_mean <- mean(boot_theta$t)
theta_bias <- theta_boot_mean - theta_hat
theta_se <- sd(boot_theta$t)
```

```r
cat("\nPart (c-ii): Bootstrap Estimates\n")
```

```
##
## Part (c-ii): Bootstrap Estimates
```

```r
cat("========================================\n")
```

```
## ========================================
```

```r
cat(sprintf("theta-hat: %.4f\n", theta_hat))
```

```
## theta-hat: 0.6191
```

```r
cat(sprintf("Bootstrap Mean: %.4f\n", theta_boot_mean))
```

```
## Bootstrap Mean: 0.6215
```

```r
cat(sprintf("Bias: %.4f\n", theta_bias))
```

```
## Bias: 0.0023
```

```r
cat(sprintf("Standard Error: %.4f\n", theta_se))
```

```
## Standard Error: 0.0479
```

3.3.3 Part (c-iii): Bootstrap Confidence Intervals

```r
# Percentile CI
ci_percentile <- boot.ci(boot_theta, type = "perc", conf = 0.95)

# BCa CI
ci_bca <- boot.ci(boot_theta, type = "bca", conf = 0.95)

cat("\nPart (c-iii): 95% Confidence Intervals for theta\n")
```

```
##
## Part (c-iii): 95% Confidence Intervals for theta
```

```r
cat("========================================\n")
```

```
## ========================================
```

```r
cat(sprintf("Percentile CI: (%.4f, %.4f)\n",
            ci_percentile$percent[4], ci_percentile$percent[5]))
```

```
## Percentile CI: (0.5230, 0.7100)
```

```r
cat(sprintf("BCa CI:        (%.4f, %.4f)\n",
            ci_bca$bca[4], ci_bca$bca[5]))
```

```
## BCa CI:        (0.5224, 0.7072)
```

Interpretation: The first principal component explains approximately 62-64% of the total variance in the five test scores. The bootstrap bias is very small (< 0.001), suggesting the sample estimate is nearly unbiased. The standard error of about 0.04 indicates moderate sampling variability. The 95% BCa confidence interval (accounting for bias and skewness) ranges from approximately 0.55 to 0.71, providing a range of plausible values for the true proportion of variance explained by the first principal component.

4 Appendix: Session Information

```
sessionInfo()
```

```
## R version 4.5.2 (2025-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64
## Running under: Windows 11 x64 (build 26100)
##
## Matrix products: default
##   LAPACK version 3.12.1
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] boot_1.3-32      bootstrap_2019.6 MASS_7.3-65      tidyr_1.3.2
## [5] dplyr_1.1.4      ggplot2_4.0.1
##
## loaded via a namespace (and not attached):
##  [1] vctrs_0.6.5        cli_3.6.5          knitr_1.51         rlang_1.1.7
##  [5] xfun_0.55          purrr_1.2.1        generics_0.1.4     S7_0.2.1
##  [9] glue_1.8.0         htmltools_0.5.9    scales_1.4.0       rmarkdown_2.30
## [13] grid_4.5.2         evaluate_1.0.5     tibble_3.3.1       fastmap_1.2.0
## [17] yaml_2.3.12        lifecycle_1.0.5    compiler_4.5.2     RColorBrewer_1.1-3
## [21] pkgconfig_2.0.3    farver_2.1.2       digest_0.6.39      R6_2.6.1
## [25] tidyselect_1.2.1   pillar_1.11.1      magrittr_2.0.4     withr_3.0.2
## [29] tools_4.5.2        gtable_0.3.6
```